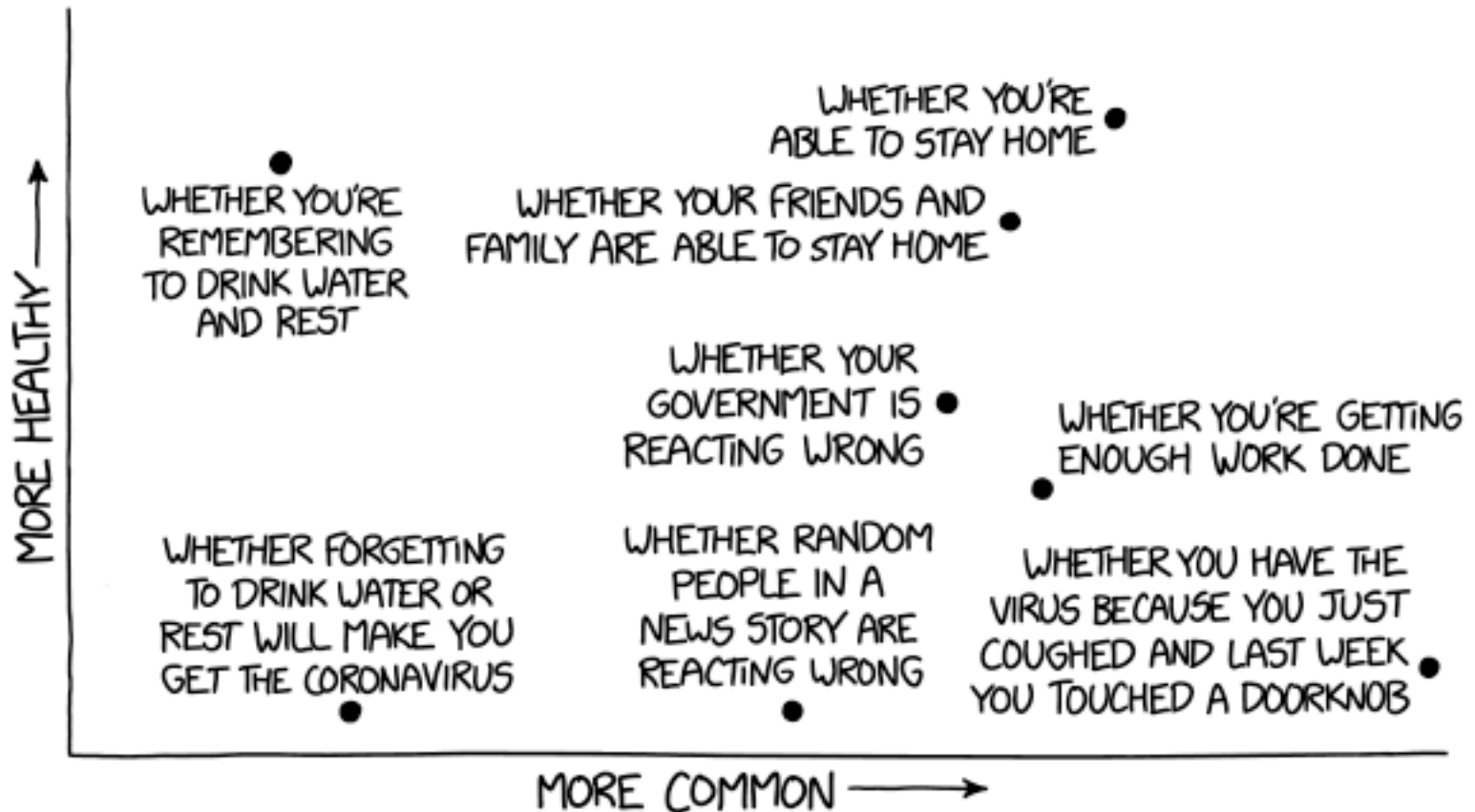
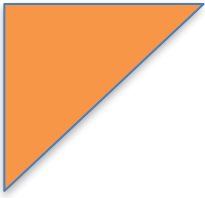


# CORONAVIRUS WORRIES



Thanks to xkcd



**COMMONWEALTH OF AUSTRALIA**

**Copyright Regulations 1969**

**WARNING**

**This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act).**

**The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.**

**Do not remove this notice.**



# L5 Common Computer Peripherals

There are a number of commonly used interface circuits which allow a computer to interface with and gather information from the outside world.

Initially we will look at parallel interfaces for inputting and outputting digital signals to/from a computer.

Digital input signals can come from switches, threshold circuits that convert an analogue value (such as light intensity and temperature) to a binary value, etc. Digital output signals can switch on lights, start motors, lock the doors in a central locking system, etc.

We will also look at two forms of serial interface used to transmit binary data.

Input and output of analogue values will be considered later.

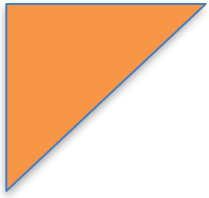


# Parallel I/O ports

One of the features that distinguishes a microprocessor from a microcontroller is that a microcontroller has built-in parallel Input/Output (I/O) lines.

In a microprocessor system parallel I/O is provided by separate parallel input/output chips such as the

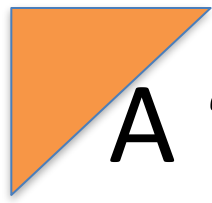
- Z8420 Parallel Input/Output Controller from Zilog,
  - I8255 Programmable Peripheral Interface from Intel
- and
- MC6821 Peripheral Interface Adapter from Motorola.



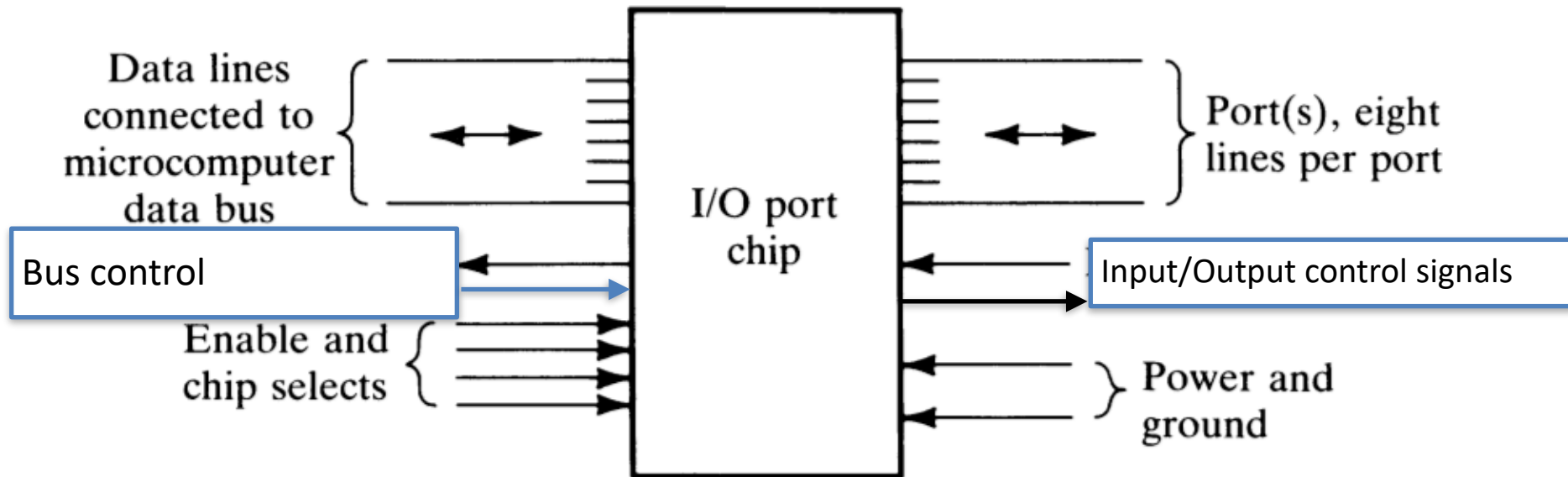
# Parallel Interface Circuits

With “ready made” interface circuits a large amount of flexibility is built into the chips and they are reconfigurable via a number of control registers. When using programmable logic the interface circuits are usually not so programmable because the “hardware” can just be modified by changing the hardware description language description.

As a simple example we will look at an historic parallel interface chip the I8255A.

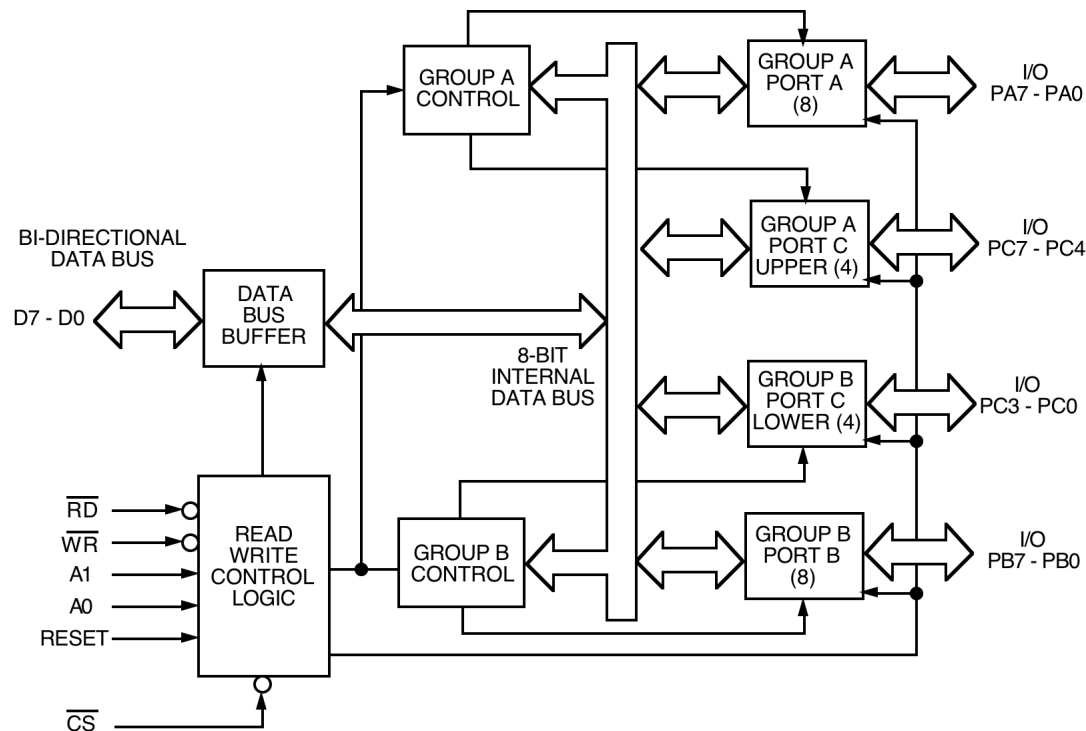


# A “generic” Parallel I/O Interface



# I8255A

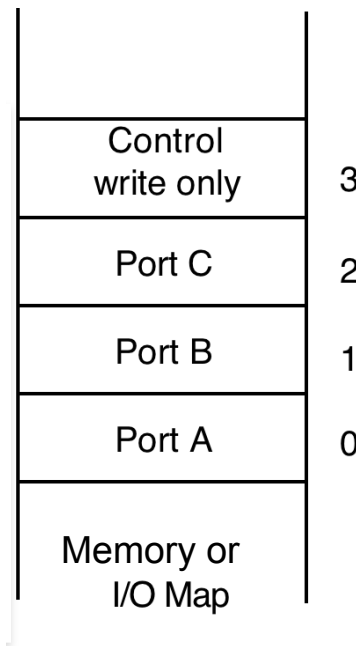
The 8255A is a general purpose parallel input/output circuit. It was used, among other things, to provide the parallel printer port on the original IBM PC. The internal structure of the chip is shown in the following diagram:





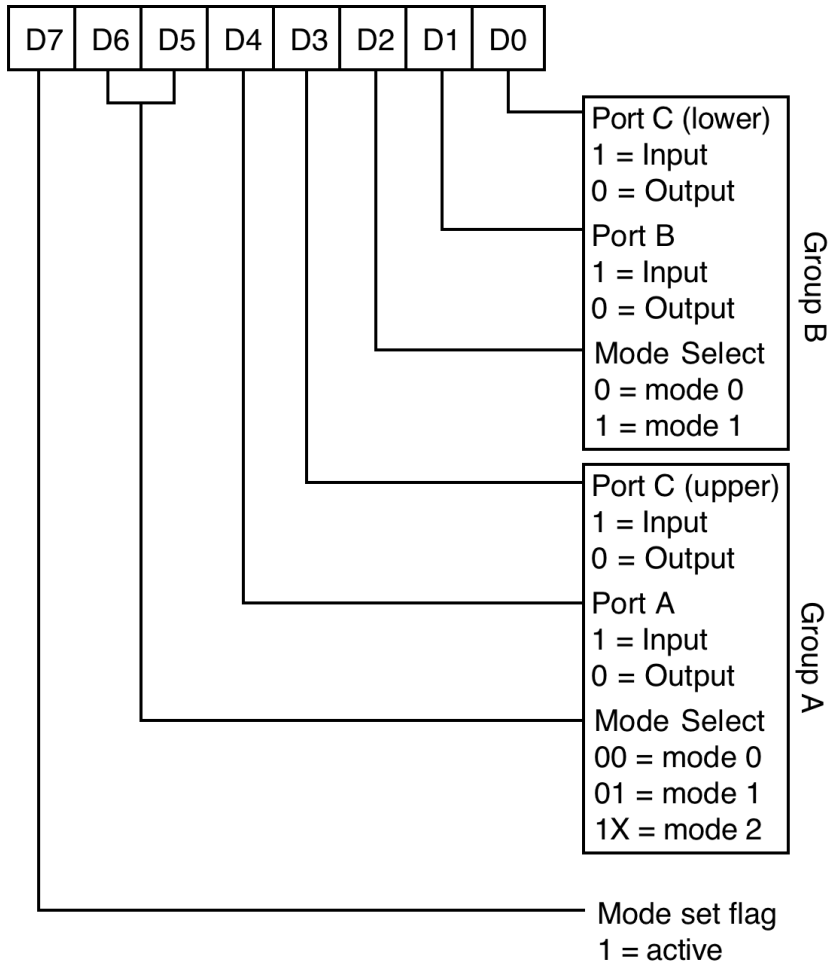
# I8255A

The computer bus interface lines are on the left hand side of the diagram and three 8-bit parallel I/O ports (ports A, B and C) are shown on the right hand side. Port C is split into two 4-bit parts, C-upper and C-lower. On power up and reset all of these ports are set to input. However, each port can be programmed as output. From the point of view of the computer the I8255A appears as four registers.





# Mode Control Word



Port A can be programmed as 8 bits input or output

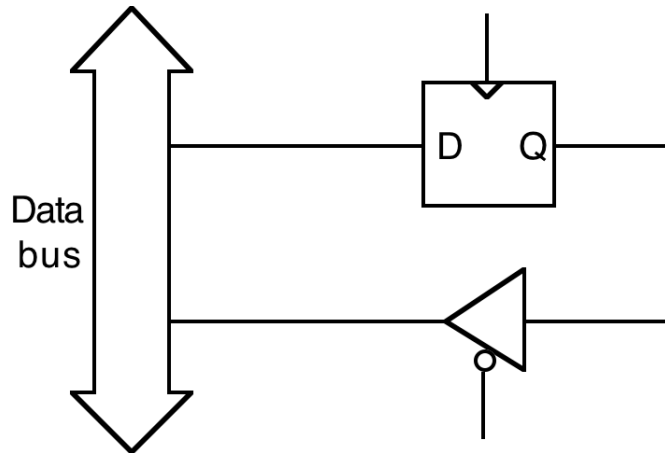
Port B can be programmed as 8 bits input or output

Port C0-3 can be programmed as 4 bits input or output

Port C4-7 can be programmed as 4 bits input or output

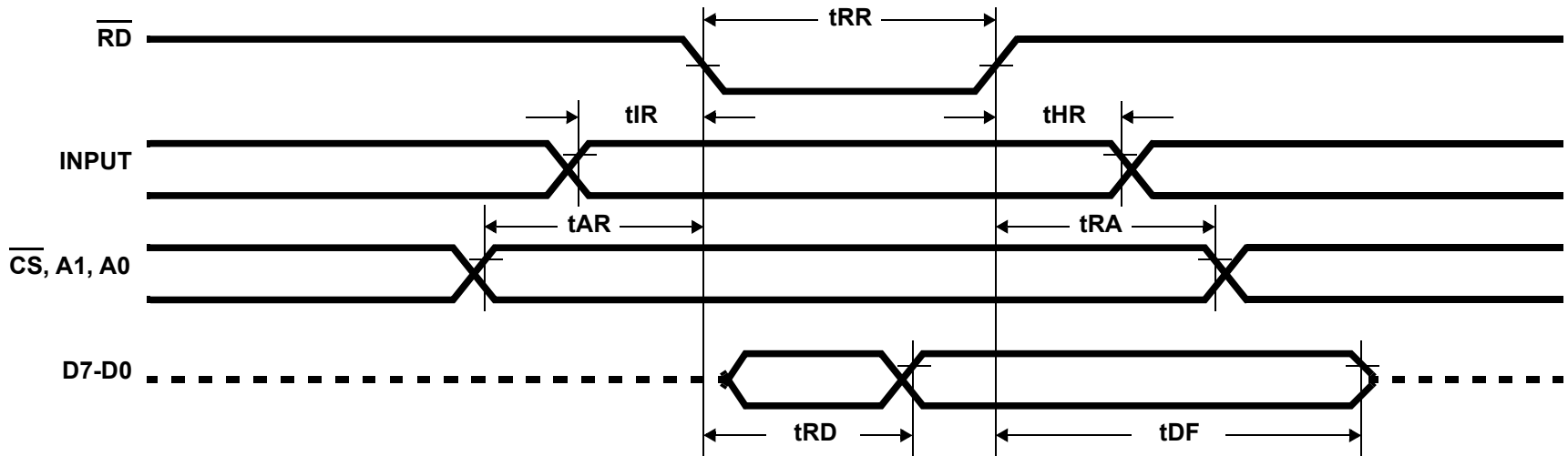
The mode control word determines how the I/O ports are configured. Bit 7 determines the mode. For Input/Output operations bit 7 is 1.

# Mode 0: Basic Input/Output



- provides 2 x 8 bit & 2 x 4 bit ports each of which can be programmed as input or output.
- output data is latched, input data is buffered.

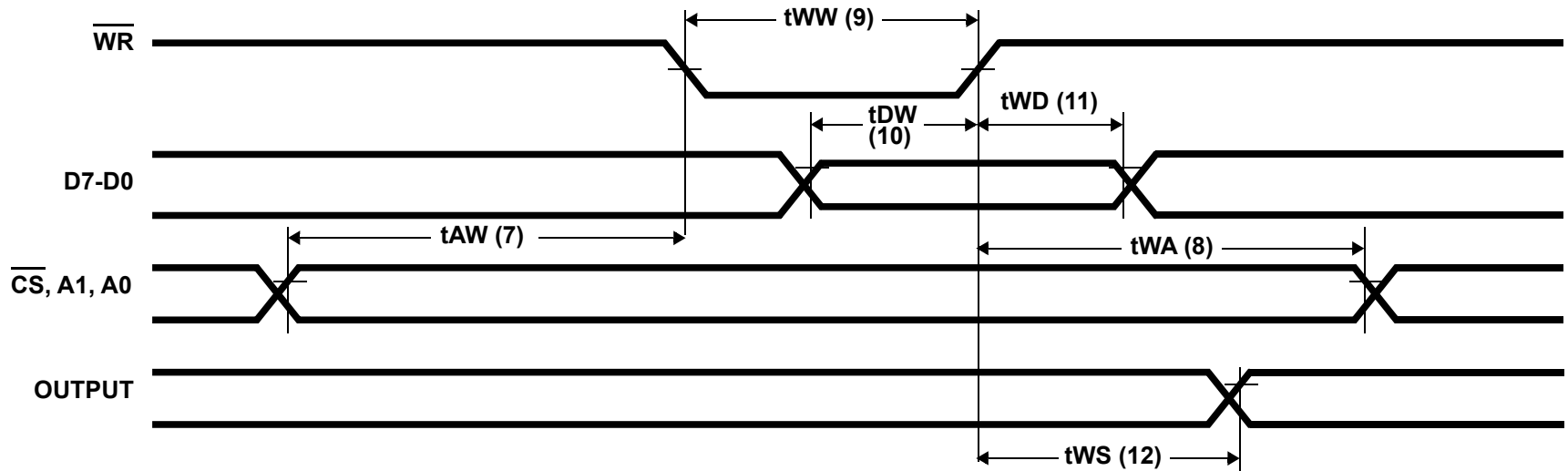
# Mode 0 (Basic Input) Timing



Essentially the Input information is buffered through the I/O port onto the Data Bus. It can be seen from the timing diagram that the Addressing and Input data must be stable before the beginning of the  $\overline{RD}^*$  pulse.

The data is buffered through the 8255A and becomes stable on the data bus for reading by the CPU before the rising edge of the  $\overline{RD}^*$  pulse.

# Mode 0 (Basic Output) Timing

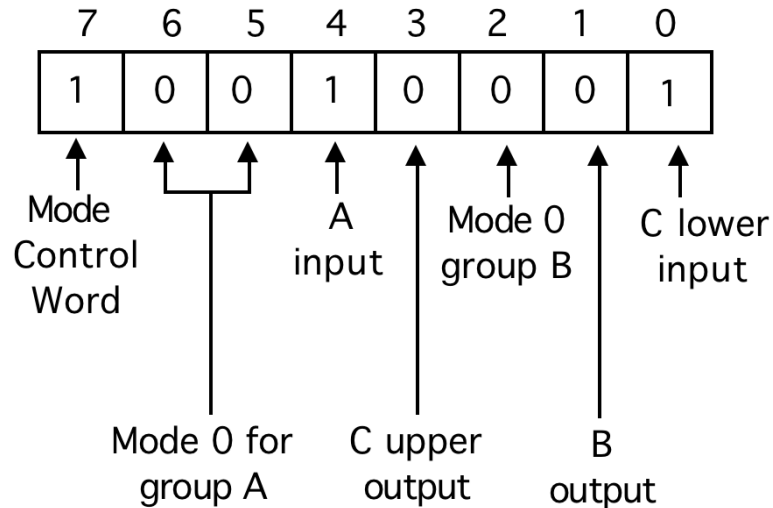


The Address bus must be stable before the beginning of the  $\overline{WR}^*$  pulse  
Output data on the data bus must then be stable before the rising edge of  $\overline{WR}^*$   
which latches the data in the 8255A.

The data then becomes stable on the the Output Port soon after.

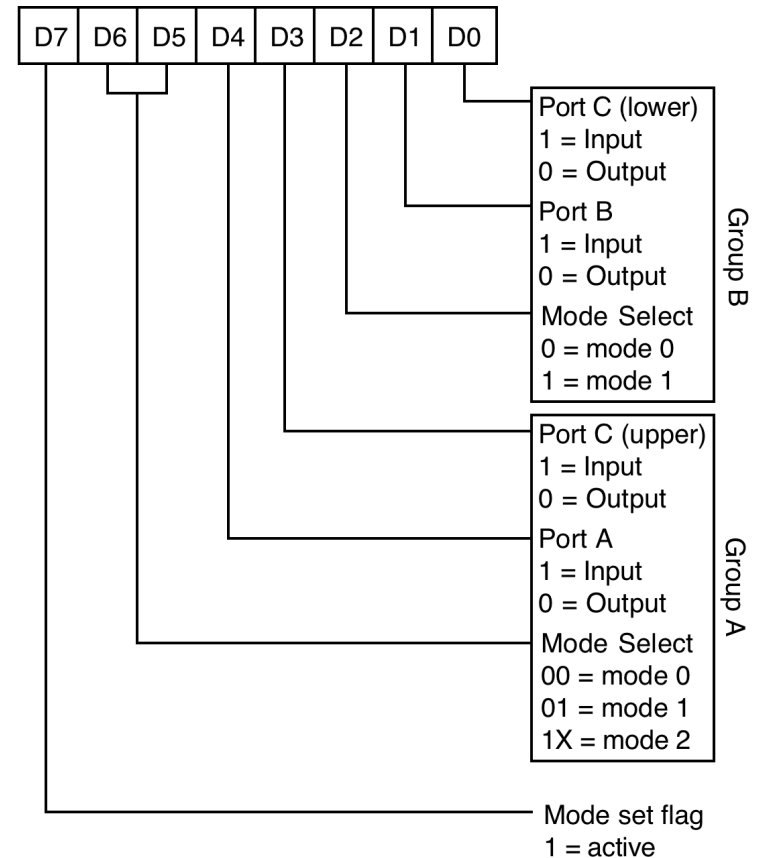
# Example

Determine the mode control word required to set Port A & Port C lower as basic input ports and Port B & Port C upper as basic output ports.



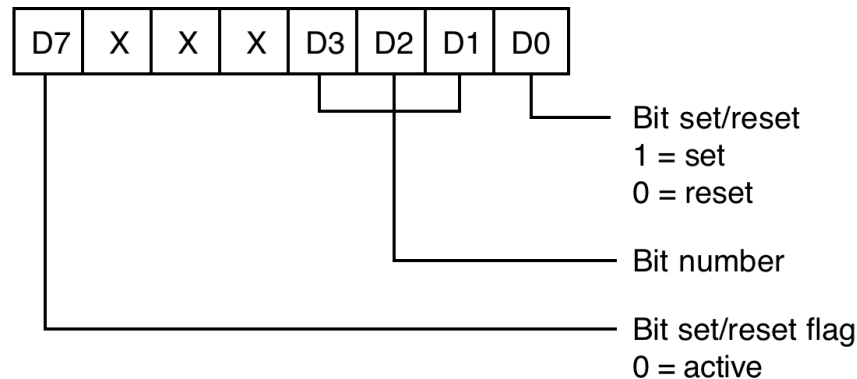
Note that:

- writing to an input port has no effect.
- reading an output port returns the last output data sent to that port or 00h if not written to after the last reset.



# Bit Set/Reset

Individual bits in Port C can be set or reset using the bit set/reset format control word.

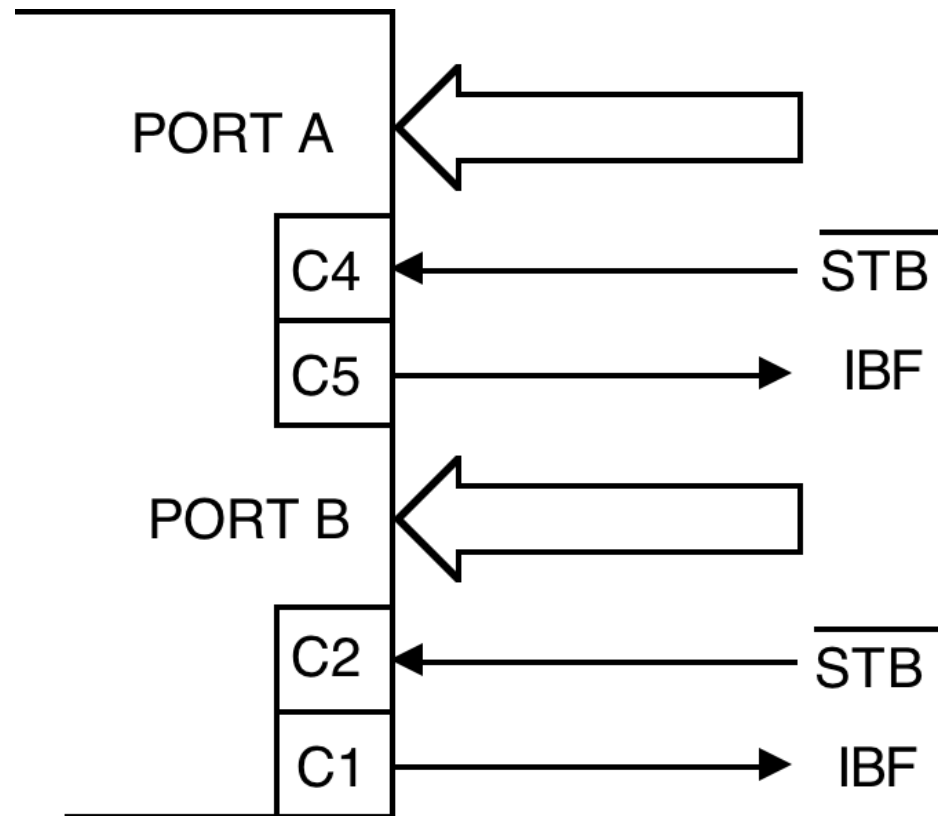


Example - to reset Port C bit 3 the following control word is written to the control register:

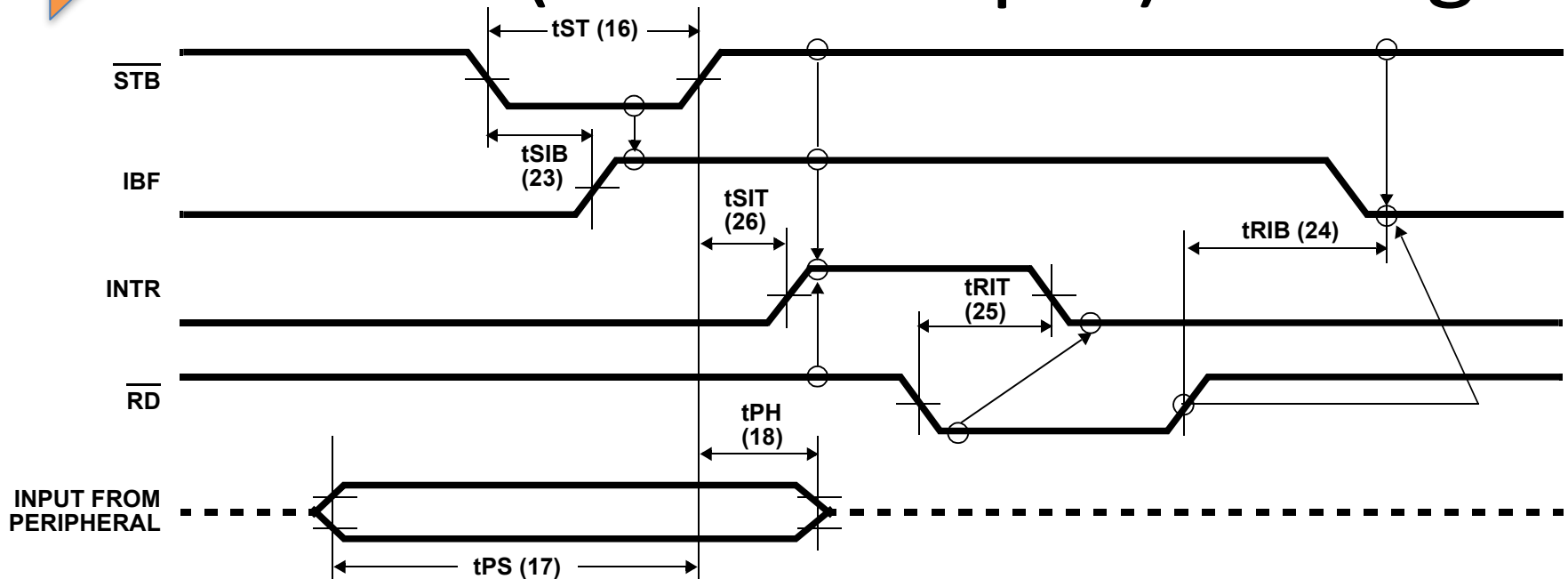
D7	X	X	X	D3	D2	D1	D0
0	X	X	X	0	1	1	0

# Mode 1: Strobed Input

In mode 0 input is only buffered and therefore a peripheral device must maintain its input until read. In mode 1 input is latched and the peripheral can remove its input before it has been read by the microprocessor.



# Mode 1 (Strobed Input) Timing



Strobed input has two phases:

Phase 1: Getting the data:

IBF becomes active after the leading edge of  $\overline{\text{STB}}^*$

The input data must be stable before the rising edge of  $\overline{\text{STB}}^*$  which latches the data.

Phase 2:

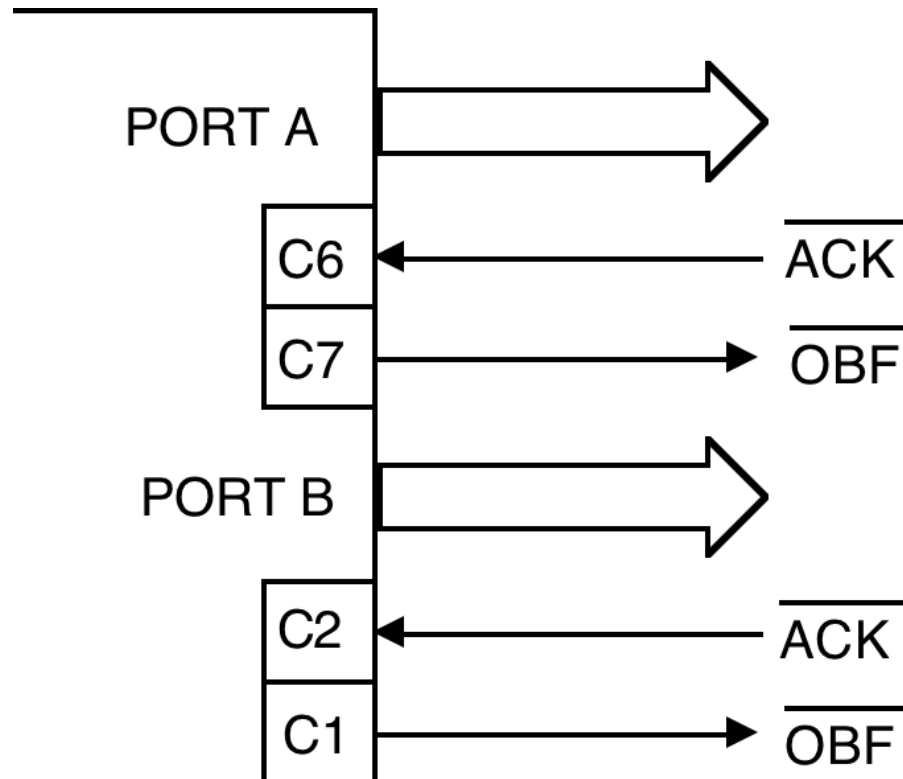
The INTR becomes active and the processor can then use  $\overline{\text{RD}}^*$  to read the data from the latches into the processor. INTR becomes inactive after the  $\overline{\text{RD}}^*$  pulse has started.

IBF becomes inactive after the rising edge of  $\overline{\text{RD}}^*$

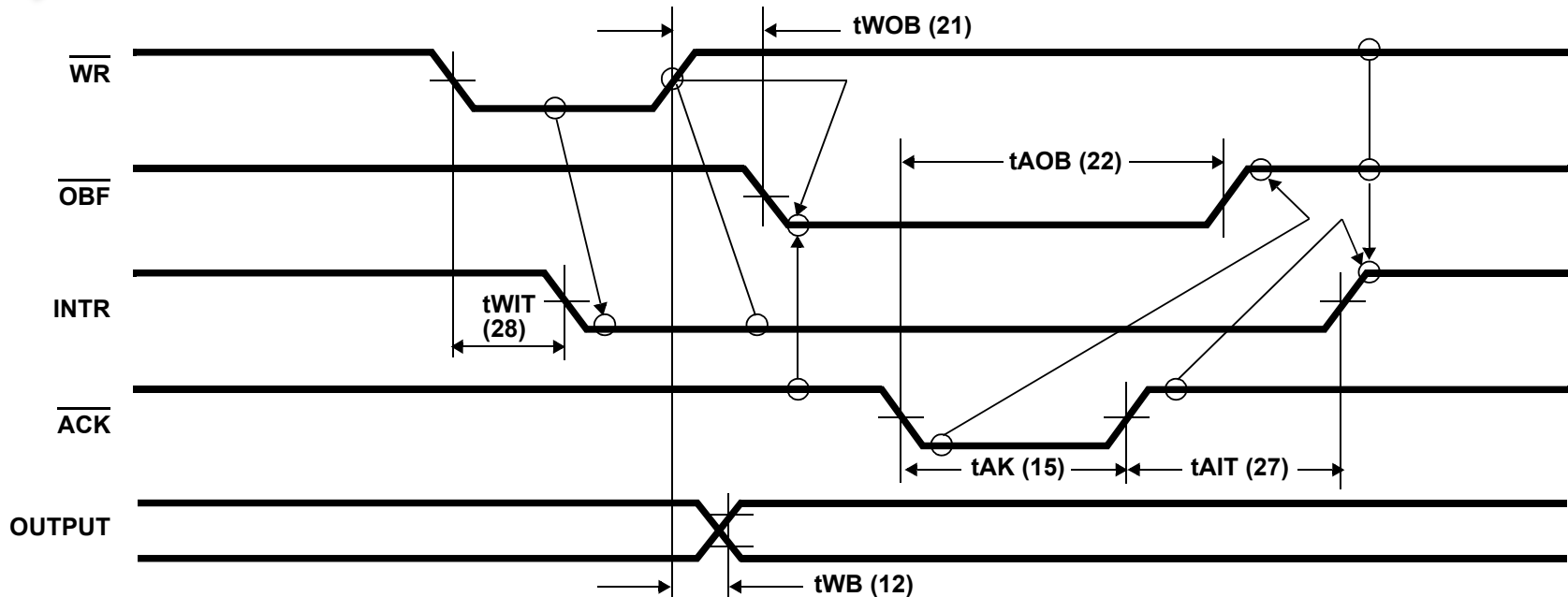


# Mode 1: Strobed Output

Strobed output provides a handshake to control the outputting of data by the I8255A. When data is written to the I8255A by the computer (WR\*) then the I8255A asserts Output Buffer Full (OBF\*). This signal remains asserted until the external circuit acknowledges reading the data by pulsing Acknowledge (ACK\*).



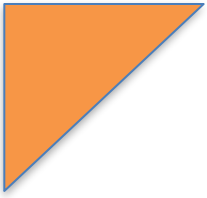
# Mode 1(Strobed Output) Timing



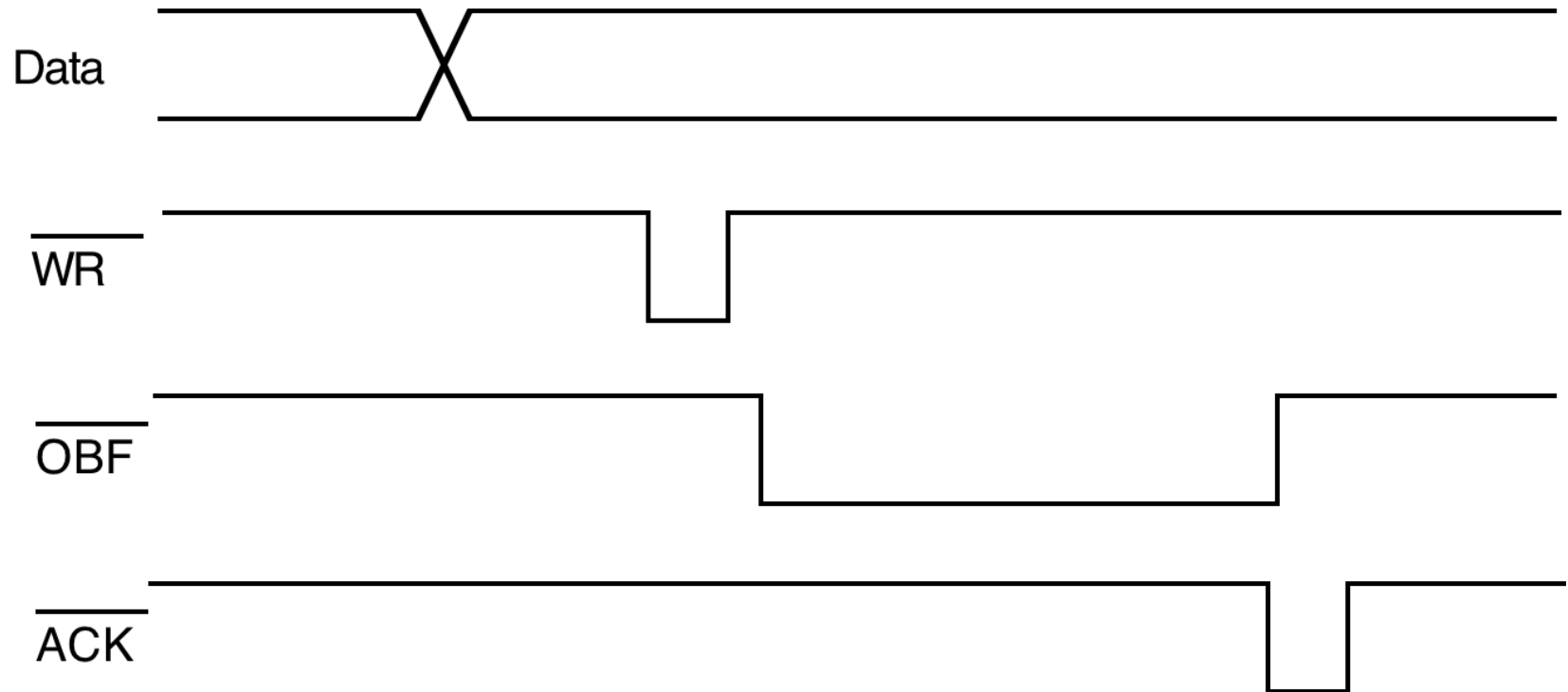
The rising edge of  $\overline{WR}^*$  latches the data into the 8255A and the data becomes valid on the Output Port soon after.

$\overline{OBF}^*$  becomes active and when the data is taken by the peripheral (The  $\overline{ACK}^*$  pulse) the  $\overline{OBF}^*$  is released.

Note: The INTR signal becomes active high after the end of the  $\overline{ACK}^*$  pulse to request new data from the processor. It becomes inactive (low) when new data is written to the 8255A

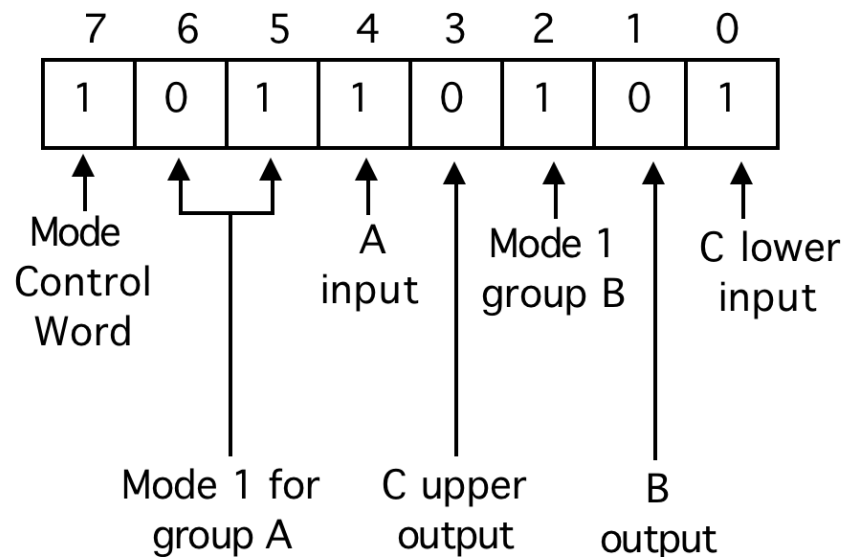


# Strobed Output Waveform



# Example

Set Port A = (Mode 1, Input), Port B = (Mode 1, Output)  
and the remaining bits of C lower as input, C upper as output.





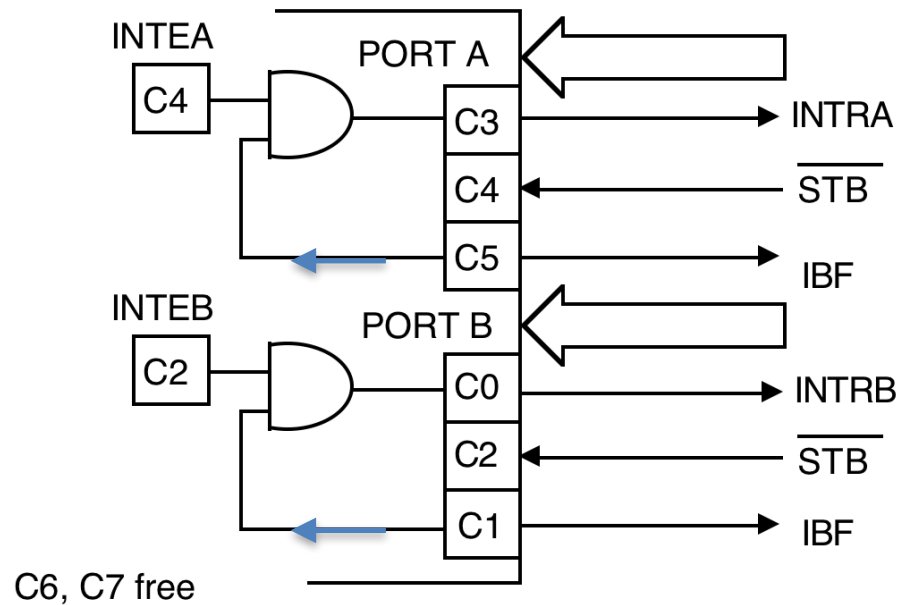
# Mode 2: Strobed Bi-Directional I/O

When initialised for mode 2 the I8255A behaves like a mode 1 input port and a mode 1 output port simultaneously. There is one significant difference - the I/O lines go high impedance when not receiving or transmitting.

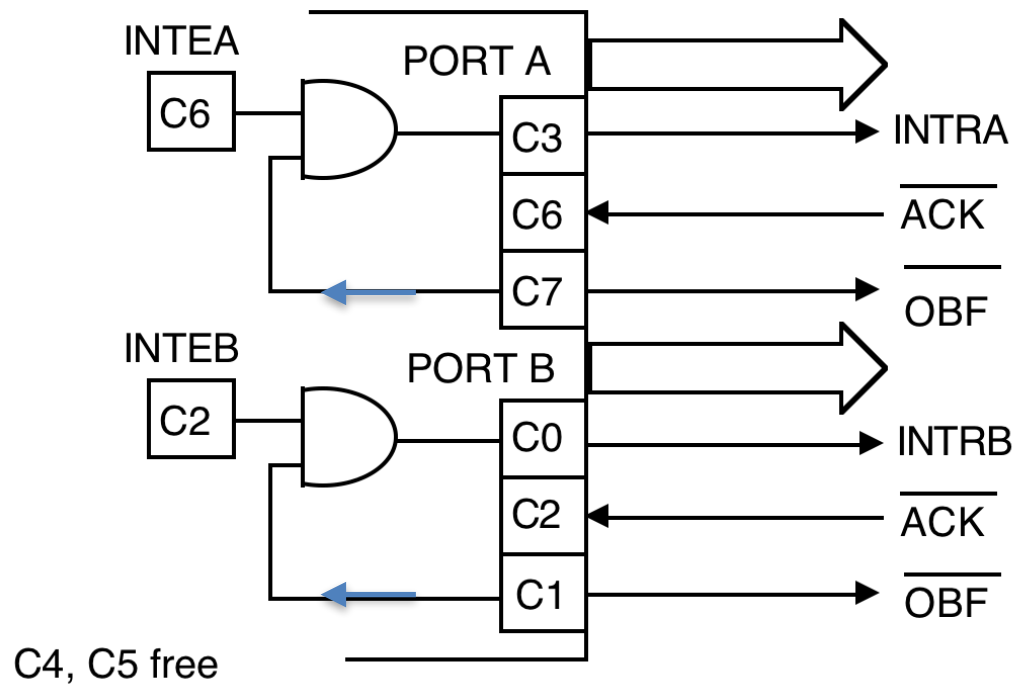
In Modes 1 and 2 a maskable interrupt enable bit is provided for each I/O port.

# Interrupts for mode 2 input

In Modes 1 and 2 a maskable interrupt enable bit is provided for each I/O port.



# Interrupts for mode 2 output



# Examples of digital Input sources

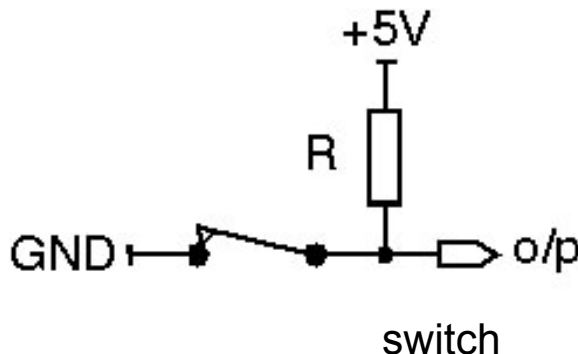
So where might digital input signals come from?

Switches can be used to indicate that a moving object such as a door, moving gantry, latch, etc., has reached a particular position. In addition, by the addition of a transducer element a switch can indicate other quantities as well as position. For instance:

A diaphragm -> air or water pressure

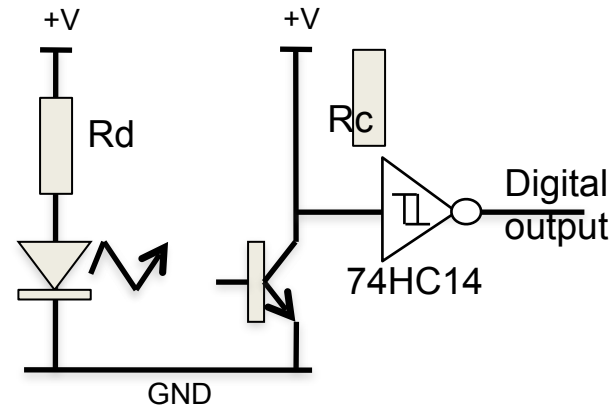
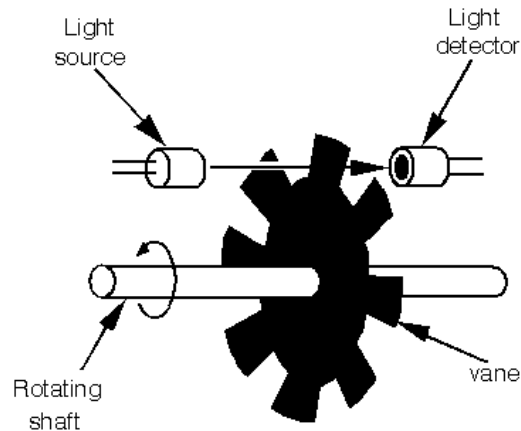
A bimetallic strip -> temperature

A deformable element -> force





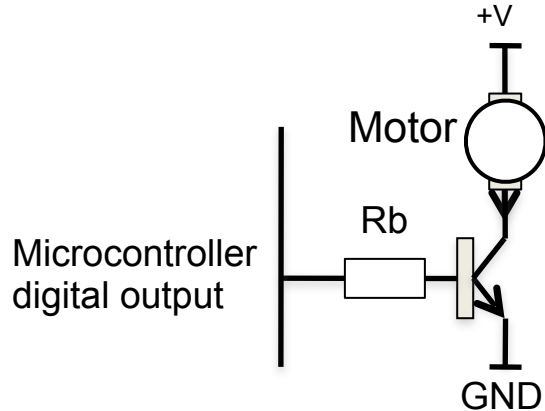
# Thresholding Analogue signals to Produce Digital Signals



- To measure a shaft rotation an LED light source shines through a slotted vane.
- As the disk rotates, light alternately falls on a phototransistor and is cut off.
- When light falls on the phototransistor it conducts and pulls the input to the 74HC14 low. When the light is cut off the transistor stops conducting and  $R_c$  pulls the input to the 74HC14 high.
- The 74HC14 inverts the logic input and its hysteresis prevents small fluctuations in the input from affecting the digital output.

# Digital Output Example

Most digital outputs from microcontrollers have a very limited current driving capability (they cannot source or sink currents larger than a few milliamps – read the manufacturer's data)



To drive a load that needs more current than the microcontroller output can provide, you can use a transistor amplifier.

Choose a transistor rated to carry the load current  $I_c$  when saturated.

Transistor base current  $I_b = n * I_c / \beta$

Where  $\beta$  is transistor current gain

$n$  is a 'safety' factor to ensure the transistor is saturated (5 is a good starting point)

$$R_b = (V_{out\_high} - V_{be}) / I_b$$

Where  $V_{be}$  is about 0.7V and

$V_{out\_high}$  is the logic high output of the microcontroller.



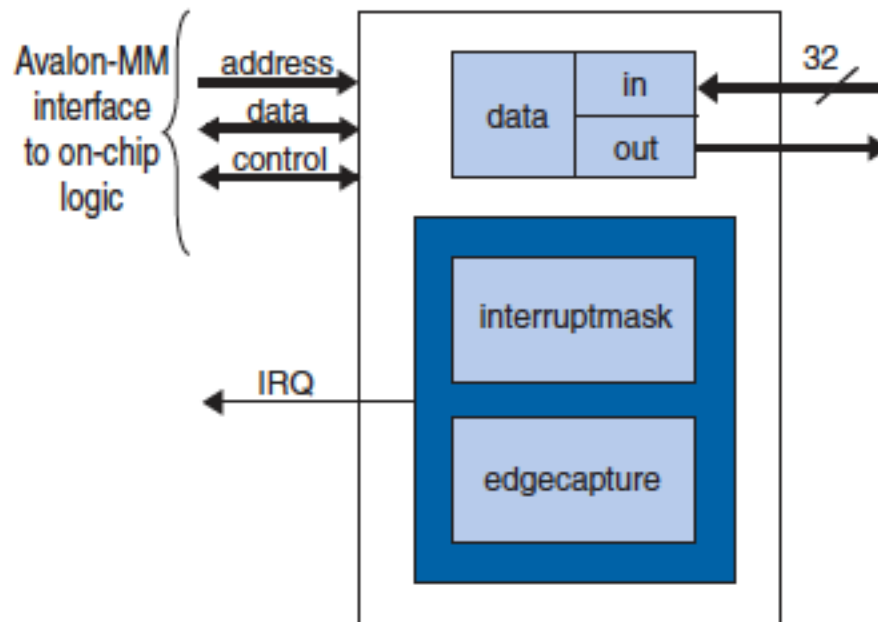
# What of modern Parallel I/O?

- Has the capability changed?
- What do you need to look for?

# Nios PIO

As an example here is details of the PIO available in the Nios system

## **PIO Core with Input Ports, Output Ports, and IRQ Support**





# Nios PIO Registers

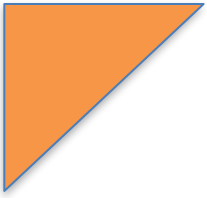
**Table 10–2. Register Map for the PIO Core**

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction (1)		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask (1)		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture (1), (2)		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				



# Engineering knowledge and examinations

- Closed book examinations:
  - You need to know the basics/generic information. Any detail needed will usually be provided.
- Open book examinations:
  - You need to know the basics/generic information. Any detail needed will usually be found in the reference material.
- Online examinations:
  - You need to know the basics/generic information. Any detail needed can be relatively easily sourced from reference material or obtained via the internet. You need to know where to look to minimise loss of time.



# Data Transmission

To transfer information between a computer and an external device a parallel connection is fast (transferring say 8 or more bits of data at the same time).

However, when the data must be transferred over a large distance providing many parallel wires is costly. In addition, there is a problem with skew between data lines limiting the speed of valid transmission.

The alternative is to transmit data serially. Thus instead of requiring 8 data lines we only require one (plus a ground return).

I KEEP HAVING NIGHTMARES THAT  
I SHOW UP AT SCHOOL, AND THEN  
SUDDENLY PANIC AS I REALIZE—

—THAT YOU'RE NAKED?

THAT I'M IN A  
CROWDED ROOM!



<https://xkcd.com/2285/>

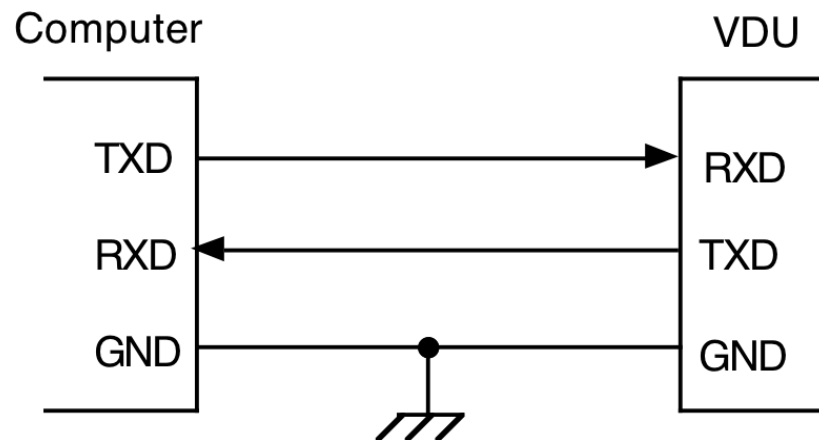


# Asynchronous Serial Comms

The transfer of information over short and long distances is becoming increasingly important in our technological society. For this reason digital communications is an essential facility for modern computer systems.

For short distances parallel transfer provides good bandwidth but over longer distances it is necessary to use the minimum number of wires.

In its “normal” form Asynchronous Serial communications uses just one wire in each direction (plus ground return).





# RS232C and RS485

- Oldest form and obsolete.
  - Signal levels +/-12volts (actually above and below +/- 3volts was valid)
  - Max speed over 12m      20kbs
- RS485
  - Differential drive so higher noise immunity and greater communications data rates. Twisted pair cable minimising interference.( +/- 200mV)
  - Max speed over 12m      35Mbs

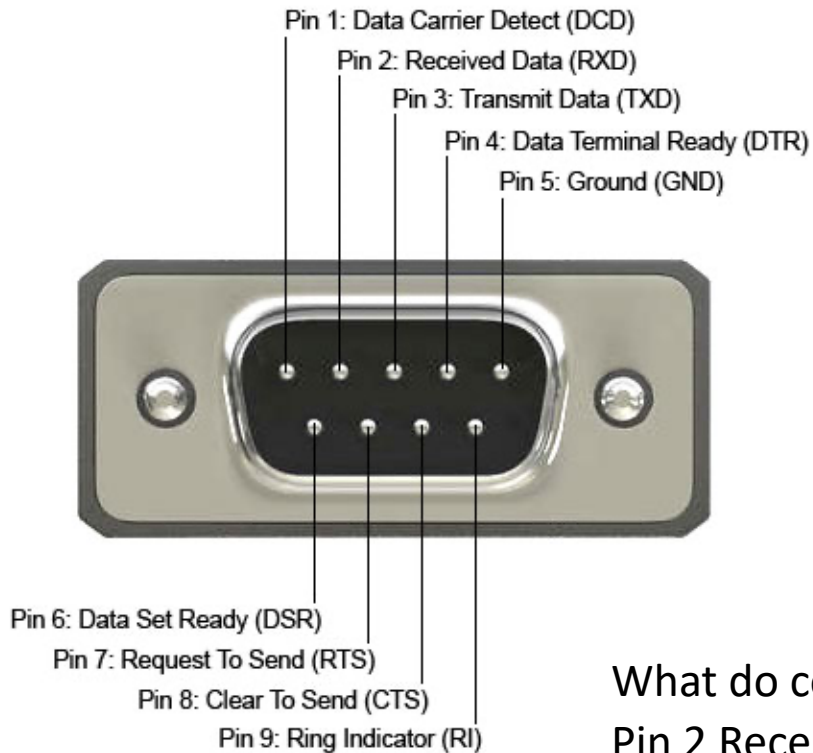


# DB25 and DB9 connectors



# DB9 Pinout

## RS232 Pinout




What do computer engineers remember from this?

Pin 2 Received data

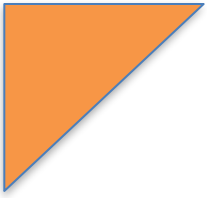
Pin 3 Transmit data

Pin 5 Ground



# What information do you need to learn here?

- The physical interface — — — -No.
  - You need to be aware of this but not learn.  
You can always look up the details.
- The format/framing of the information — — Yes.
  - You need to learn and understand how the data is formatted and sent.



# Synchronisation

Start and stop bits ensure that there is always a HI to LO transition at the start of each frame.

This is detected at the receiver and used to synchronise the receiver clock to the transmitter clock.

Synchronisation is only required to be maintained for one frame because the receiver resynchronises at the start of the next frame.

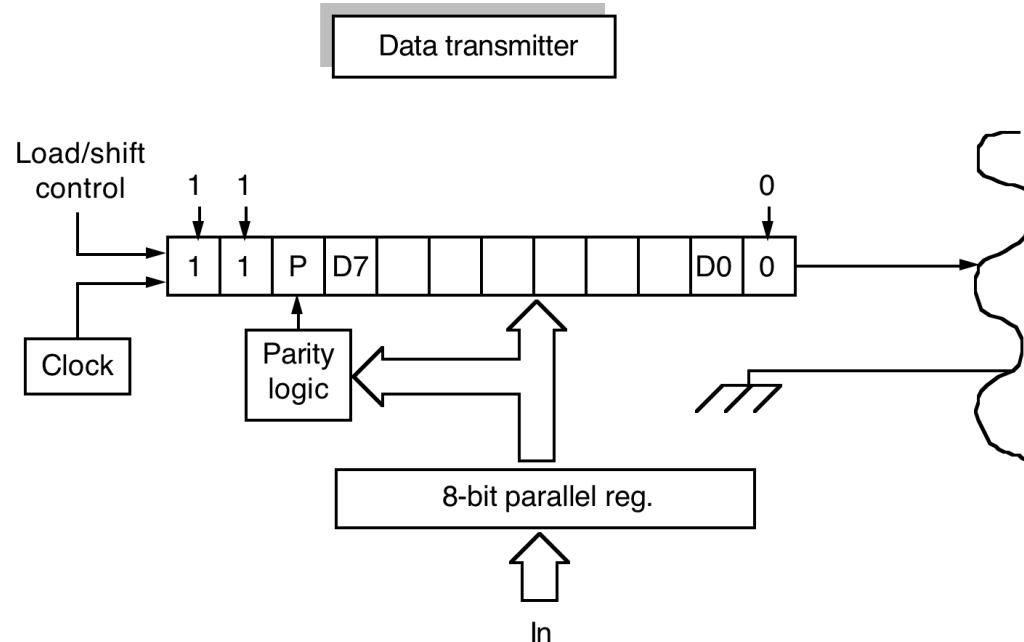
Typically the local receiver clock is running at 16 times the data clock rate. After detecting the synchronising signal edge the received value is rechecked after 8 cycles of the local clock and if the right signal is present data sampling starts 16 cycles later and continues every 16th cycle until the frame is complete.

# Asynchronous Transmission

Parallel data from a microcomputer at the transmitting end is parallel loaded into a register.

When the previous frame has finished being shifted out of the shift-register then the new data is parallel-loaded into the shift register together with start, parity and stop bits.

The resulting frame is then shifted out one bit at a time.

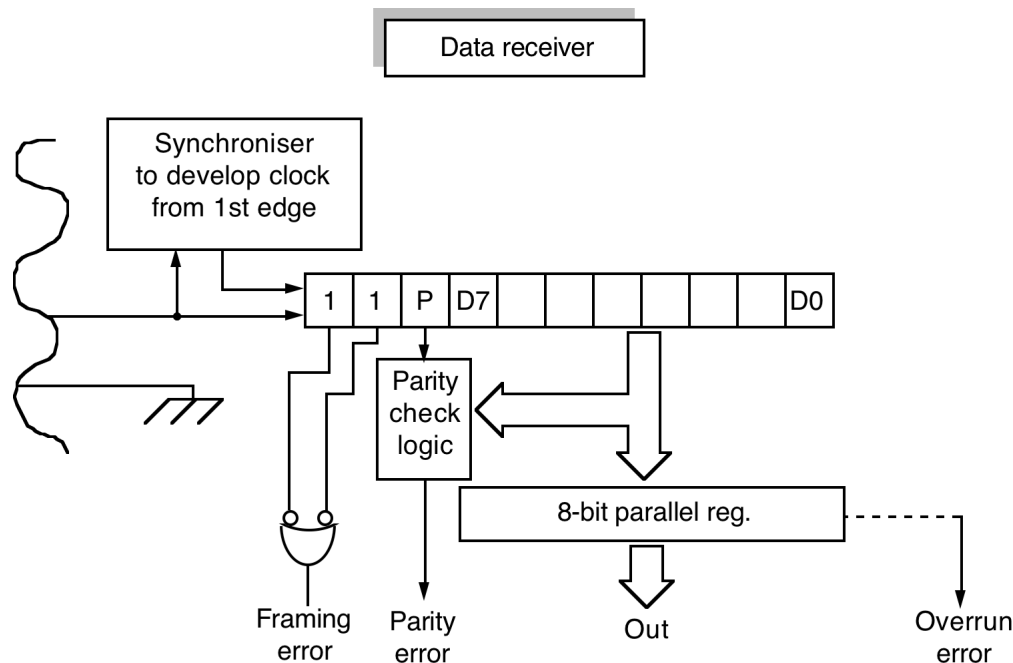


# Asynchronous Reception

Start and stop bits ensure that there is always a HI to LO transition at the start of each frame.

This is detected at the receiver and used to synchronise the receiver clock to the transmitter clock.

Synchronisation is only required for one frame because the receiver resynchronises at the start of the next frame.

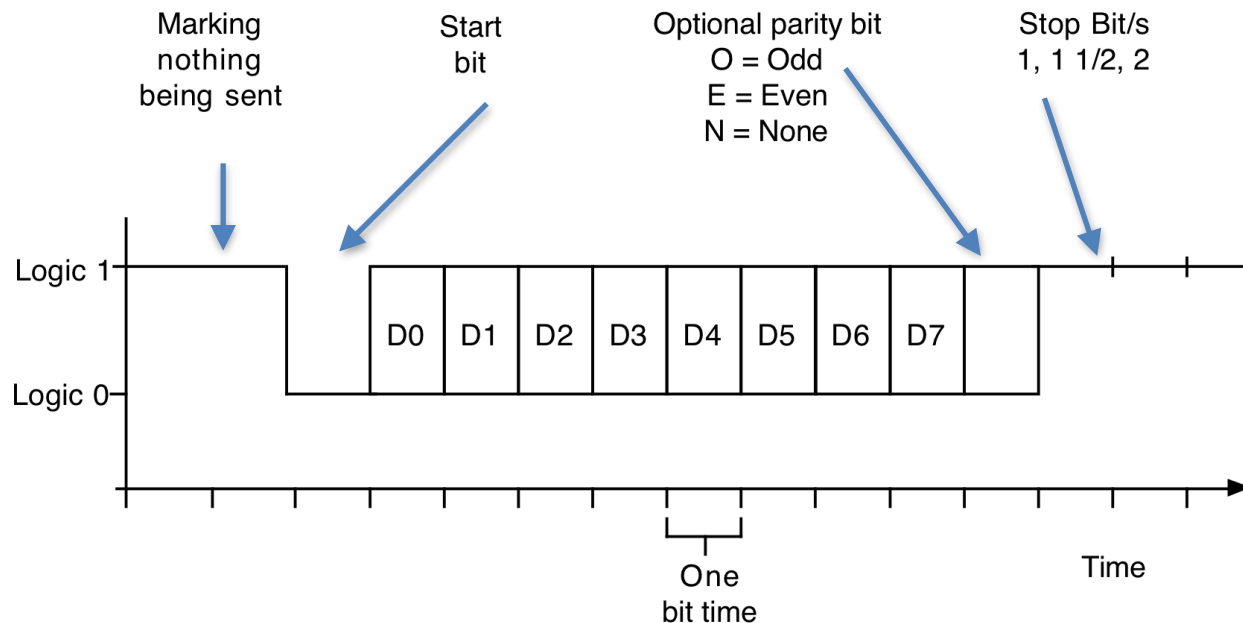




# A Frame

Information is sent down the single communications path as a sequence of bits organised into frames.

One frame usually transfers one byte or one character and contains features which allow the receiver to work out where one frame ends and the next starts.



BE CAREFUL - Time increases to the right of the diagram so the first bit to be transmitted is bit D0



# Synchronisation

Synchronisation between transmitter and receiver is performed using start and stop bits to ensure that there is at least one HI -> LO -> HI transition for every character transmitted.

If there were no start and stop bits consider the bit pattern that would be transmitted if the data was:

- 1) many groups of 0000000b with even parity, or
- 2) many groups of 1111111b with even parity.

In each of these cases there would be no way of telling where one byte finished and the next started



# Baud Rate

The speed with which information is transferred on a serial link is called the Baud rate. Baud rate is defined as:

$$\text{Baud rate} = 1/(\text{bit time})$$

Commonly used baud rates include 75, 150, 300, 600, 1200, .... ,9600, 19200 baud.

Because of the additional start bit, stop bit/s and parity bit the actual transfer of information in bits/sec is less than the Baud rate.



# Example

How many characters/second could be transmitted at 9600 baud if the characters were represented in 7 bits with one parity bit and 2 stop bits?

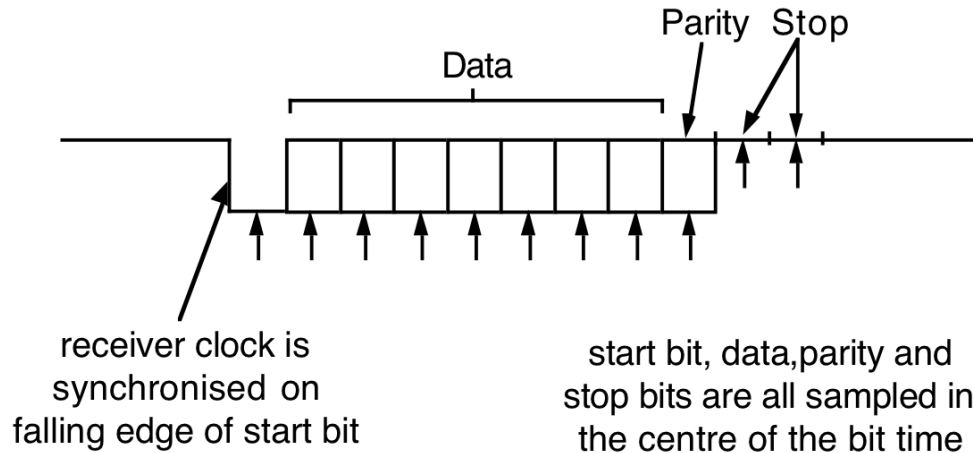
$$\begin{aligned}\text{character time} &= (\text{total number of bits/char}) \times \text{bit time} \\ &= (1 \text{ start bit} + 7 \text{ data bits} + 1 \text{ parity} + 2 \text{ stop bits}) / (9600) \\ &= 1.146 \text{ ms}\end{aligned}$$

$$\begin{aligned}\text{characters/sec.} &= 1/(\text{character time}) \\ &= 872.7 \text{ char/sec (= 6109 data bits/sec)}\end{aligned}$$

Note that this example is using 7 data bits.  
What is the answer if we are transferring 8 data bits?

# Clock Accuracy

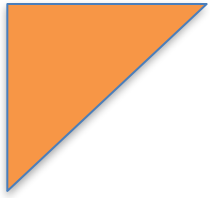
How accurately must the receiver and transmitter clocks be matched so that data is received correctly?



Characters will be received correctly if the sampling point has drifted less than 1/2 a bit time between the point where the receiver clock was synchronised and the last stop bit. Therefore in the above example (7 data bits, parity and 2 stop bits) the error must be less than 1/2 a bit time in a period of 10.5 bit times.

$$\text{Combined error in Tx and Rx} < = \pm 0.5 / 10.5 * 100\% = \pm 4.76\%$$

If this error is distributed equally between transmitter and receiver an accuracy of better than  $\pm 2.38\%$  would be required for correct operation.



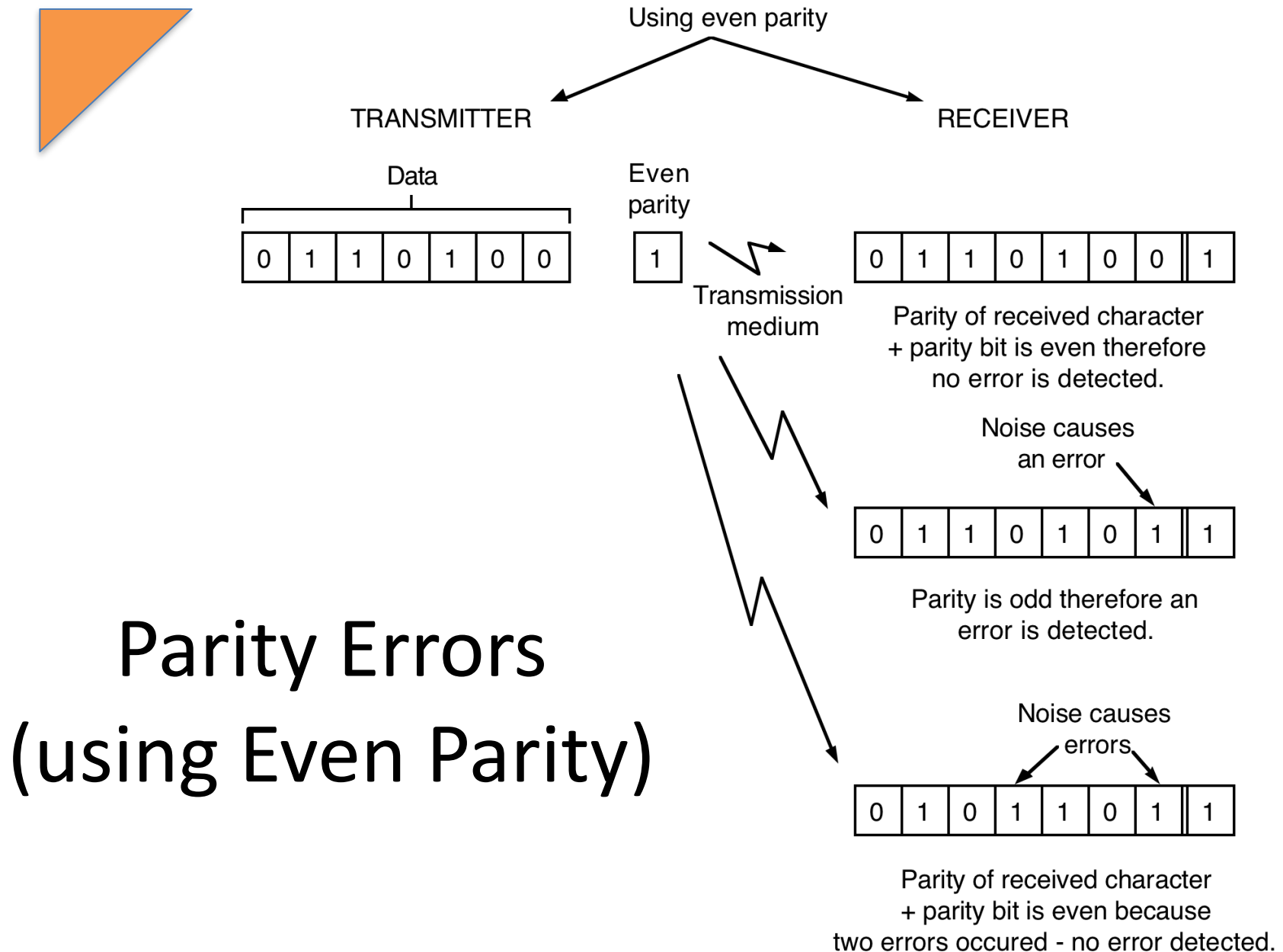
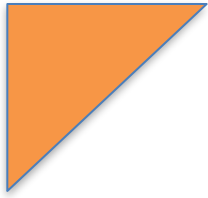
# Parity

Parity is a simple form of error detection attached to each transmitted character (in this case).

Odd parity - means that there are an odd number of 1's in the (data + parity).

Even parity - means that there are an even number of 1's in the (data + parity).

The transmitter and receiver must be configured to use the same kind of parity.





# Example

Data 0110100 is to be transferred over a serial link using even parity to detect errors.

There are three 1's in the data and therefore the parity bit is a 1 to make the total (data + parity bit) even.

When the message is received without error there are four 1's in the (data + parity bit) and the receiver reports no parity error.

With a noisy transmission path one 0 is misinterpreted as a 1. The parity of the received message is now odd and an error is reported.

However, if two (or any even number) of errors occurs the receiver will not detect an error.





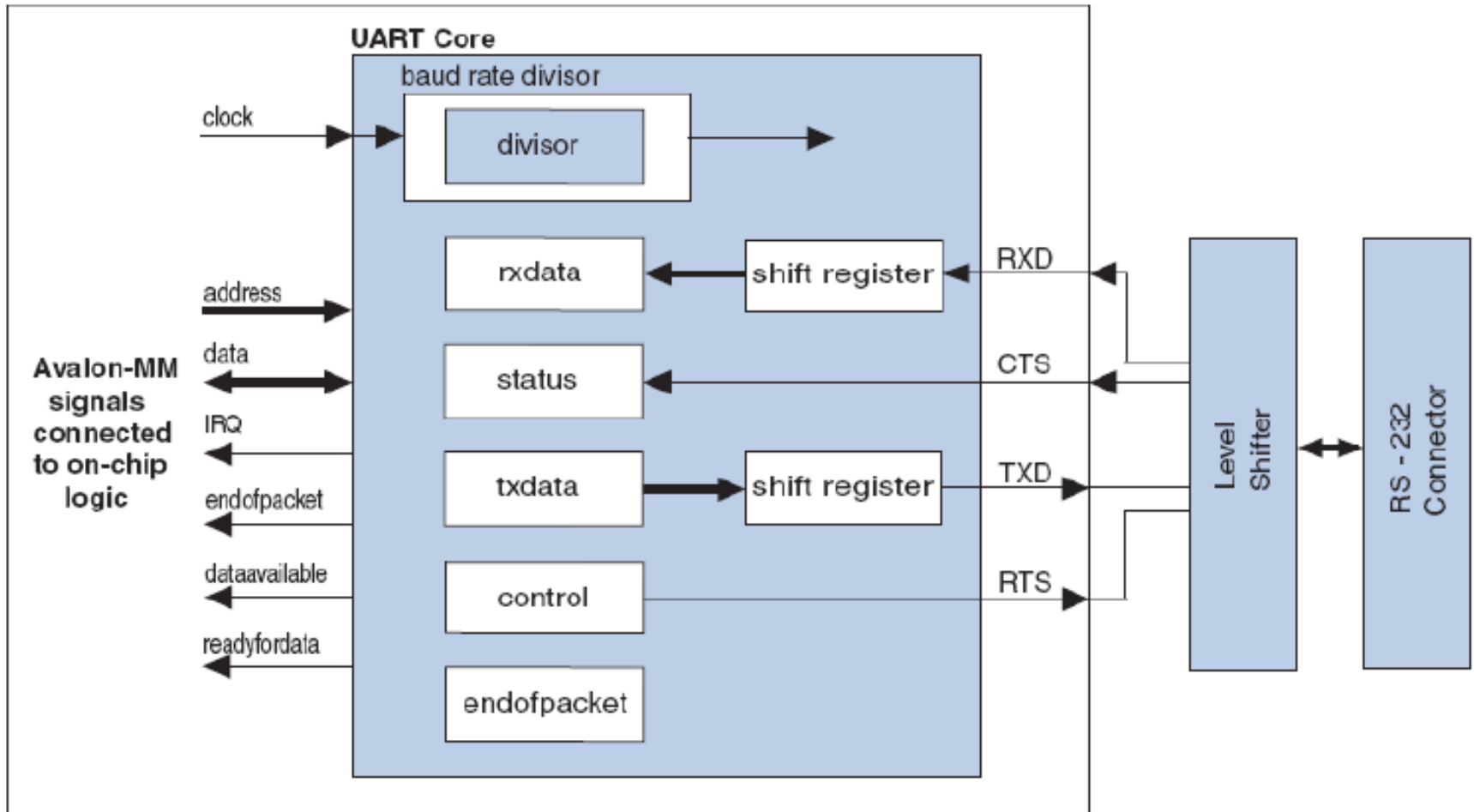
# Hardware Errors

The receiver reports any errors that it detects. They are usually:

- 1) Parity error (caused by excessive noise)
- 2) Framing error - stop bit/s is/are not high indicating that the alignment of transmitted and received data frames does not match. Can be caused by noise being interpreted as a start bit or noise corrupting a stop bit. Mismatch of baud rates between transmitter and receiver will also produce a framing error.
- 3) Overrun error - data has been lost at the receiver. The receiver buffer was full. The computer did not remove this character before another character arrived and overwrote it.

# Nios RS232 UART

Altera FPGA



# UART Registers

**Table 1. UART Register Map**

A2..A0	Register Name	R/W	Description/Register Bits															
			15	...	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	rxdata	RO									RxData							
1	txdata	WO									TxData							
2	status (1)	RW			eop	cts	dcts	–	e (2)	rrdy	trdy	tmt	toe	roe	brk	fe	pe	
3	control	RW			ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe	
4	divisor	RW	Baud Rate Divisor (optional)															
5	endofpacket	RW									End-packet value							



# UART Status

***Table 2. status Register Bits***

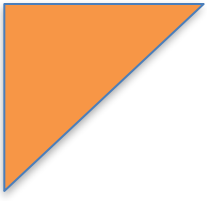
Bit Number	Bit Name	Description
0	pe	Parity error
1	fe	Framing error
2	brk	Break detect
3	roe	Receive overrun error
4	toe	Transmit overrun error
5	tmt	Transmit empty
6	trdy	Transmit ready
7	rrdy	Receive char ready
8	e	Exception
10	dcts	Change in clear to send (CTS) signal
11	cts	CTS signal
12	eop	End of packet encountered



# UART Control

***Table 3. control Register Bits***

Bit Number	Bit Name	Description
0	ipe	Enable interrupt for a parity error
1	ife	Enable interrupt for a framing error
2	ibrk	Enable interrupt for a break detect
3	iroe	Enable interrupt for a receiver overrun error
4	itoe	Enable interrupt for a transmitter overrun error
5	itmt	Enable interrupt for a transmitter shift register empty
6	itrdy	Enable interrupt for a transmission ready
7	irrdy	Enable interrupt for a read ready
8	ie	Enable interrupt for an exception
9	trbk	Transmit break
10	idcts	Enable interrupt for a change in CTS signal
11	rts	Request to send (RTS) signal
12	ieop	Enable interrupt for an end of packet encountered



# UART Programming

```
#include <altera_avalon_uart_regs.h>
```

```
IOWR(GP_UART_BASE,1,dat);  
GP_UART_data=IORD(GP_UART_BASE,0);
```

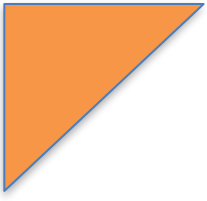
## Examples:

```
/* Write the next input_request[] character to the txdata register  
(Register 1).*/
```

```
    IOWR(GP_UART_BASE, 1, input_request[index]);
```

```
/* A receive character is ready, so read it from the rxdata register  
(Register 0).*/
```

```
    incoming_character = IORD(GP_UART_BASE, 0);
```



Common Computer Peripherals (part 3)

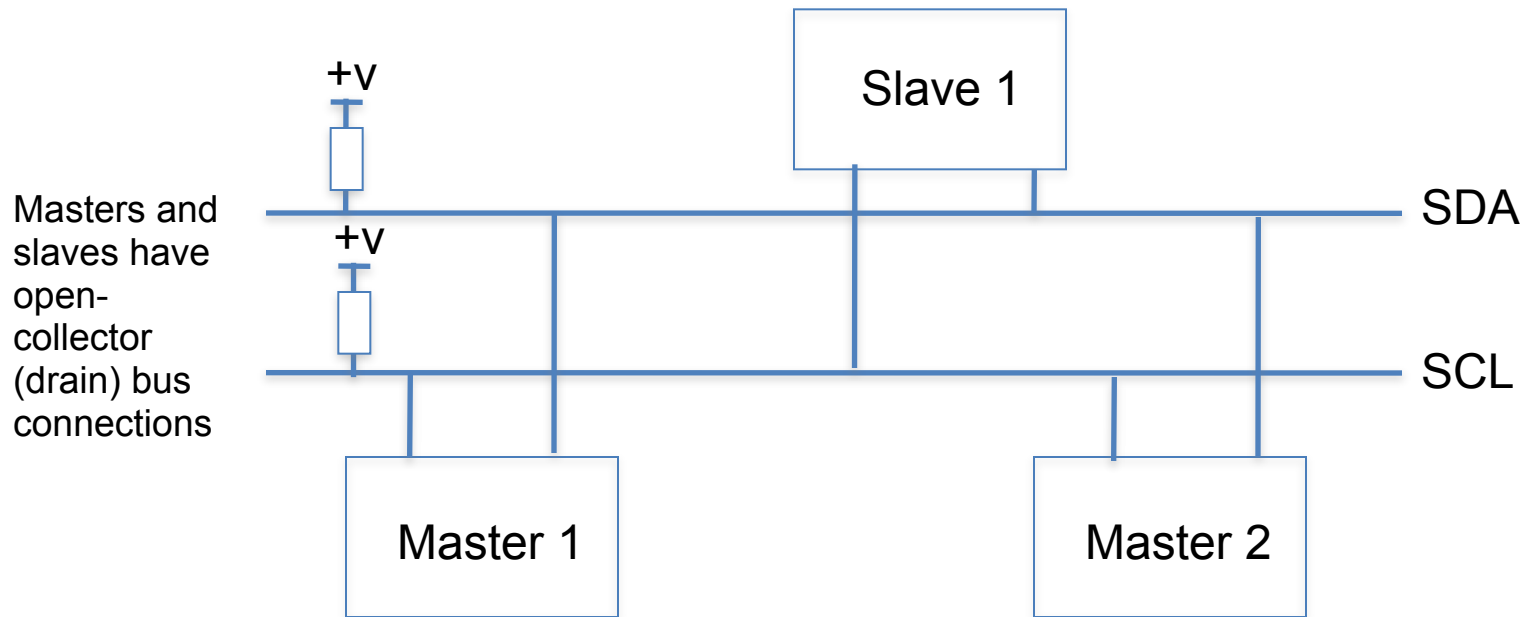
Clive Maynard

# I<sup>2</sup>C BUS

# I<sup>2</sup>C Bus

The I<sup>2</sup>C is a popular, low-cost, moderate speed (~100kBytes/sec.) and relatively simple method of data transfer to connect microcontrollers with peripheral devices. It uses only two bus lines, serial data (SDA) and serial clock (SCL).

Masters can request a response from slaves and slaves can only respond.







# Multi-master

Any device connected to I<sup>2</sup>C bus which wants to act as a master provides the **clock signal**.  
So I<sup>2</sup>C is a **synchronous** serial protocol.

Each master must monitor the bus while transmitting to make sure that it is not interfering with another master.

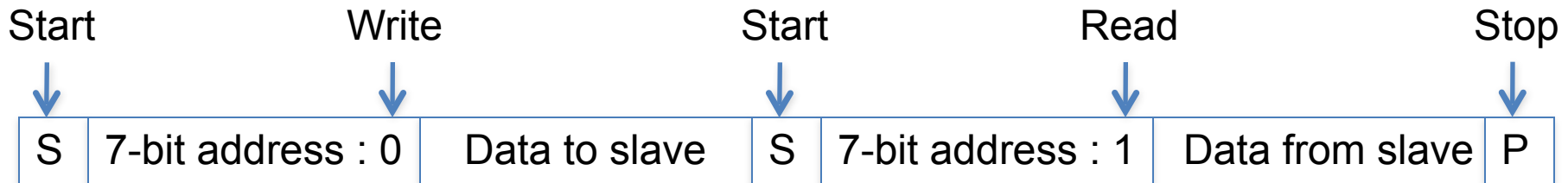
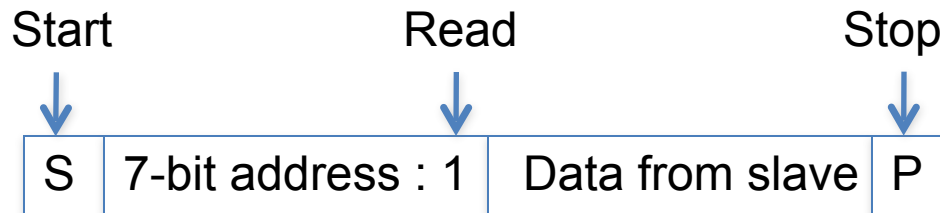
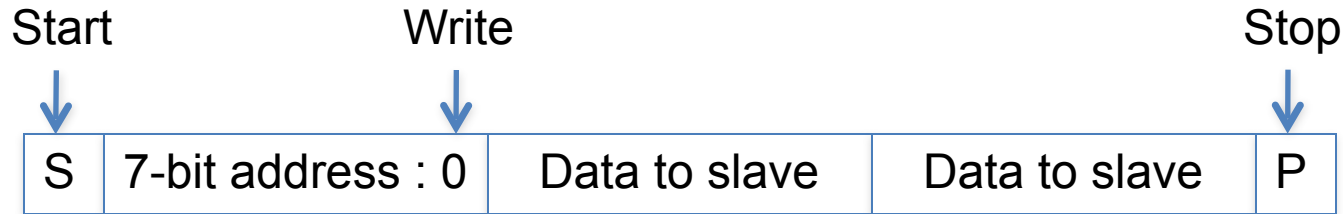
In a connected system all I<sup>2</sup>C devices have a unique 7-bit address (10-bit extended addressing scheme). Address 0000000b is reserved for a broadcast message to the whole system.



Data bits are transferred on the 0 -> 1 transitions on SCL.

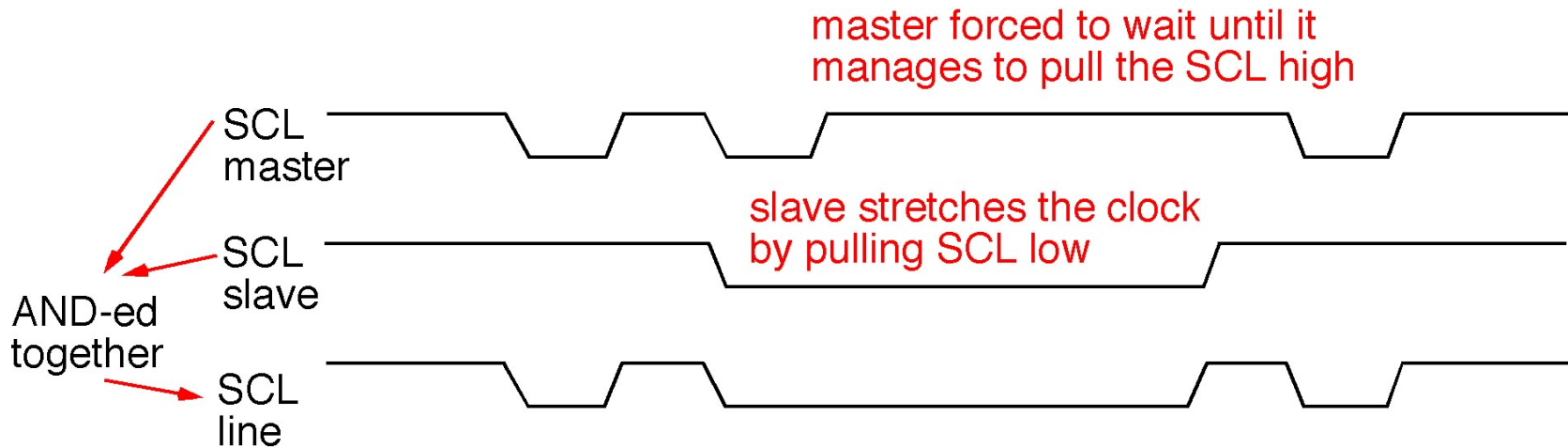
To allow the receiving device to control SDA the transmitting device lets the SDA line float.

# Examples of Bus Transactions



# Clock Stretching

If an addressed slave device is slow and is not ready to receive new data then it can slow down the master by holding down the clock signal (SCL) until it is ready for more data. The master will detect that it is not able to raise the clock line and hence will wait.



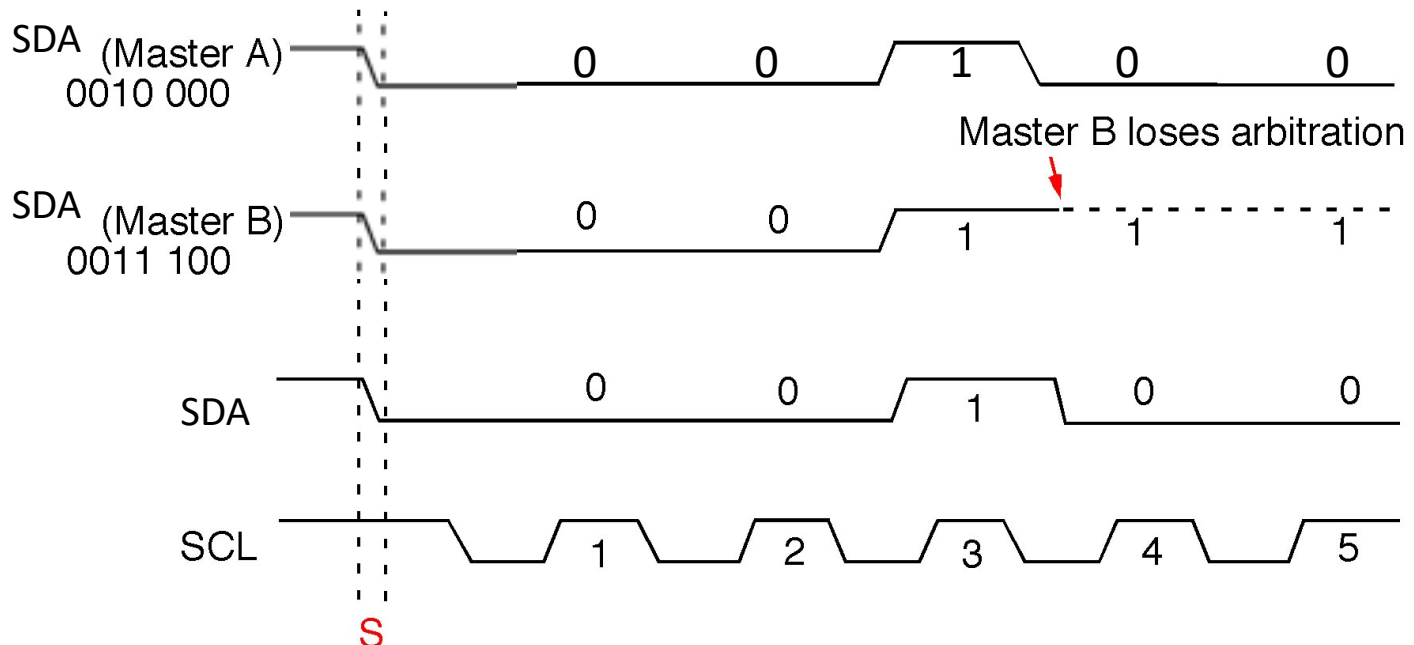
# Bus Arbitration

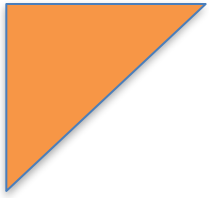
The I<sup>2</sup>C bus can support multiple master devices which can result in a problem if more than one master tries to gain control of the bus at the same time.

This problem is addressed by an arbitration scheme.

Each transmitter checks the bus to see if it is the value that it is transmitting.

If it is not then there is a conflict and the transmitter whose value is not appearing on the SDA line drops out.





# OUTSIDE THE BOX(5)

**Imagine a computer without a stack!!!**

How would you manage to have subroutines?

Can you conceive of a technique to have a functional Jump to Subroutine instruction (JSR) and probably more importantly a way to get back to where the program came from?

As a clue to a solution the process need not be reentrant.  
Only one call to a subroutine would be active at any one time!!!



# Next

- No matter how fast a computer executes its instructions many users will want it to operate quicker.
- Speed can be increased by increasing the system clock frequency. For the same microprocessor design there is a limit to the maximum clock rate directly related to the propagation delay of the logic
- The next section looks at some simple ways of increasing computer processing speed by modifying the system architecture, not by speeding up the logic